



Orchestration

Avec Apache Airflow

JEREMY GROS

#Data Engineer (Ippon depuis 10/2021)

#AgenceMarseille

#AWS #CICD #Airflow #Kubernetes #Kafka ...





SOMMAIRE

PARTIE 01 20'

Introduction
Concepts de base
Utiliser Airflow
Avancées

PARTIE 02 2H30

Setup
Présentation du sujet
TPs

PARTIE 03 10'

Conclusion
ROTI

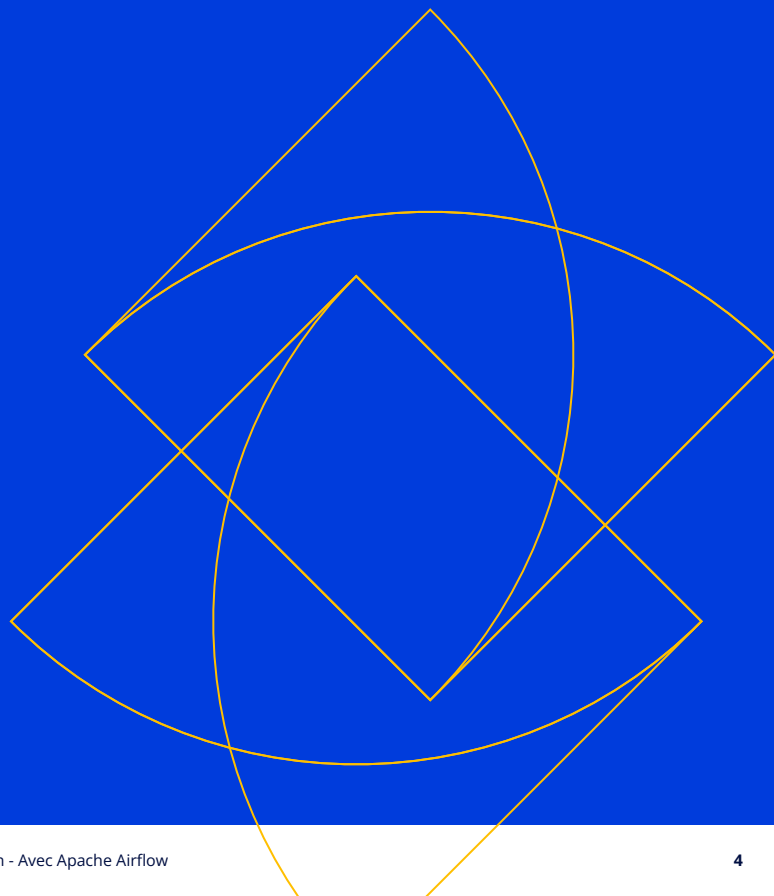


01 — Introduction

01 — Qu'est-ce que l'orchestration ?

02 — Pourquoi est-ce important ?

03 — Qu'est-ce qu'Airflow ?



Qu'est-ce que l'orchestration ?

Définition

L'orchestration est le processus de planification, de coordination de gestion des tâches et des workflows d'un système informatique.

Cela peut inclure des tâches telles que la planification de tâches répétitives, la gestion des dépendances entre les tâches et la surveillance des tâches en cours d'exécution.

Objectif

Garantir que les tâches s'exécutent de manière efficace et fiable.

=> améliorer la qualité des services, réduire les coûts et accélérer les délais de mise en production.



Pourquoi est-ce important ?

- Planification des tâches de manière efficace -> en définissant des dépendances et en planifiant les tâches en fonction des besoins.
- Surveillance des tâches en cours d'exécution et gérer les erreurs de manière efficace.
- Automatisation des tâches répétitives -> réduction des erreurs et des délais.
- Gestion des ressources de manière efficace en allouant les ressources en fonction des besoins.



Qu'est-ce qu'Airflow ?

- Système open source pour la planification et l'orchestration de workflows.
- Permet de créer, planifier et surveiller des workflows en utilisant des Directed Acyclic Graphs (DAGs).
- Développé par AirBnB / maintenu par la communauté Apache.
- Beaucoup d'options et de fonctionnalités :
 - planification : périodique, sur événement et à l'aide de dépendances;
 - surveillance : journaux détaillés, alertes, console web pour suivre l'état des tâches;
 - intégration : énormément de connecteurs avec d'autres outils et technologies;
 - gestion des erreurs : gestion des retries, gestion des alertes.



01 — Concepts de base d'Airflow

01 — DAGs (Directed Acyclic Graphs)

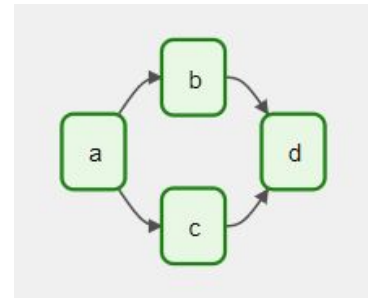
02 — Tâches

03 — Opérateurs

04 — Capteurs



DAGs (Directed Acyclic Graphs)



- Bases de tous les workflows dans Airflow
- Décrit les workflows en utilisant des graphes orientés acycliques
- Les tâches sont des noeuds du graphe et les dépendances entre les tâches sont les arcs
- Il n'est pas possible de créer des cycles au sein d'un workflow

Exemple de DAG :

```
from airflow import DAG
from datetime import datetime, timedelta

default_args = {
    'owner': 'igros@ippon.fr', 'depends_on_past': False, 'retries': 1, 'retry_delay': timedelta(minutes=5)
}

dag = DAG(
    'my_dag_name', description='A simple tutorial DAG',
    start_date=datetime(2023, 1, 1), default_args=default_args, catchup=False, schedule='@daily'
)
```



Tâches

- Étapes individuelles dans les DAGs
- Elles peuvent être des :
 - opérations de déplacement de fichiers;
 - des opérations SQL;
 - des opérations Python;
 - des appels de service CLOUD;
 - etc.
- Il existe deux types de tâche dans Airflow :
 - Opérateurs : BashOperator, PythonOperator etc..
 - Capteurs : FileSensor, S3KeySensor, ExternalTaskSensor etc..

```
task = BashOperator(  
    task_id='my_task_name',  
    bash_command='echo "Hello World!"',  
    dag=dag  
)  
  
sensor = S3KeySensor(  
    task_id='s3_file_sensor',  
    bucket_key='path/to/file.csv',  
    bucket_name='my_bucket',  
    s3_conn_id='s3_conn',  
    poke_interval=30,  
    dag=dag  
)
```



Opérateurs

- Actions / Traitements à effectuer dans le DAG
- Ils peuvent être des :
 - opérations de déplacement de fichiers
 - des opérations de SQL
 - des opérations de Python
 - des appels de service CLOUD
 - etc.
- Sur les traitements longs, il est possible de faire de l'asynchrone : Exécuter une tâche sans attendre la fin et utiliser un "capteur" pour attendre la fin de l'exécution.



Capteurs

- Opérateur particulier : Attendre qu'une condition soit validée
- Exemples :
 - FileSensor => Attend la présence d'un fichier
 - S3KeySensor => Attend la présence d'une clé dans S3
 - ExternalTaskSensor => Attend un ou plusieurs états d'une tâche airflow d'un dag
- Deux modes :
 - **poke** : Le capteur est démarré sur un "worker" et reste tant qu'il n'est pas terminé
 - **reschedule** : Le capteur est démarré sur le worker, vérifie la condition et se met dans un état "up_for_reschedule" si celle-ci n'est pas validée



01 — Utiliser Airflow

01 — Comment installer Airflow ?

02 — Architecture airflow

02 — Utilisation de la console web Airflow

03 — Comment créer et démarrer des DAGs ?

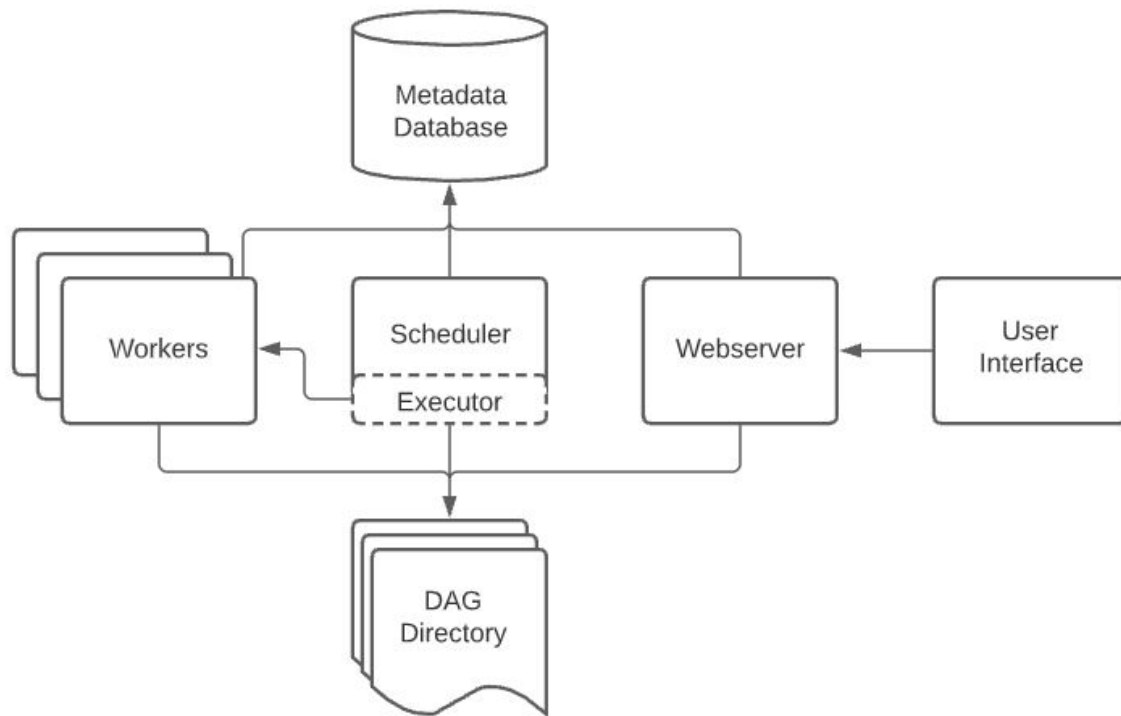


Comment installer airflow ?


- Local :
 - Standalone
 - Docker / Docker-compose
- Production ready :
 - Kubernetes : déploiement manuel, Helm Chart
 - Services SaaS
 - AWS : Managed Workflow for Apache Airflow (MWAA)
 - Astronomer : Astro
 - GCP : Google Cloud Composer



Architecture Airflow



Utilisation de la console web Airflow

 Airflow

DAGs

Security

Browse

Admin

Docs

21:11 UTC

RH

DAGs

All 26Active 10Paused 16

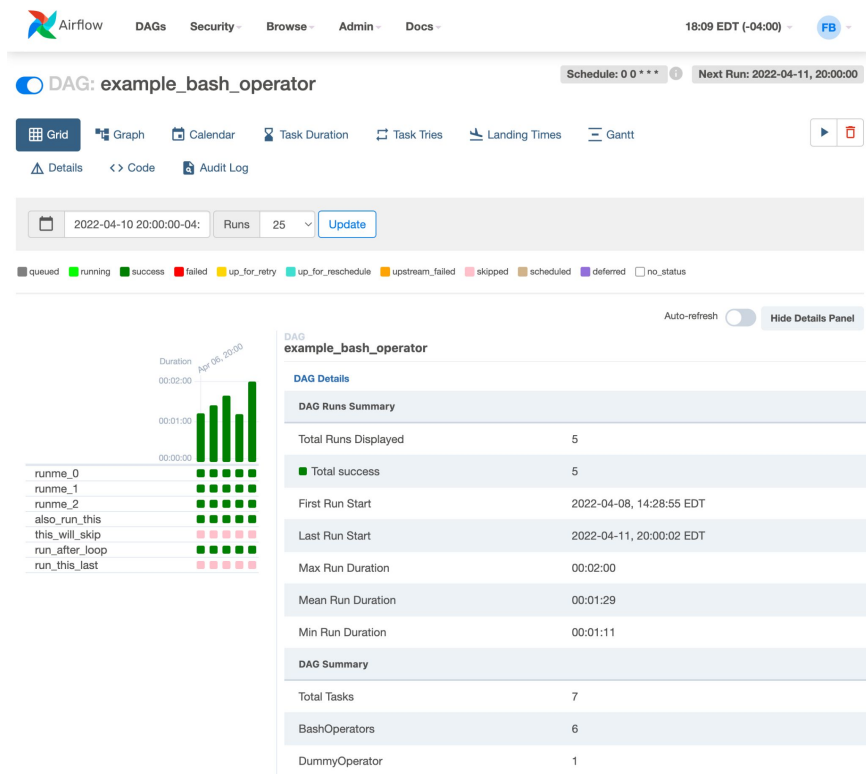
Filter DAGs by tag

Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
<input checked="" type="checkbox"/> example_bash_operator example example2	airflow	2	0 0 *	2020-10-26, 21:08:11	6	▶ ↺ 🗑️	...
<input checked="" type="checkbox"/> example_branch_dop_operator_v3 example	airflow		* / 1 *			▶ ↺ 🗑️	...
<input type="checkbox"/> example_branch_operator example example2	airflow	1	@daily	2020-10-23, 14:09:17	11	▶ ↺ 🗑️	...
<input checked="" type="checkbox"/> example_complex example example2 example3	airflow	1	None	2020-10-26, 21:08:04	37	▶ ↺ 🗑️	...
<input checked="" type="checkbox"/> example_external_task_marker_child	airflow	1	None	2020-10-26, 21:07:33	2	▶ ↺ 🗑️	...
<input checked="" type="checkbox"/> example_external_task_marker_parent	airflow	1	None	2020-10-26, 21:08:34	1	▶ ↺ 🗑️	...
<input checked="" type="checkbox"/> example_kubernetes_executor example example2	airflow		None			▶ ↺ 🗑️	...
<input checked="" type="checkbox"/> example_kubernetes_executor_config example3	airflow	1	None	2020-10-26, 21:07:40	5	▶ ↺ 🗑️	...
<input checked="" type="checkbox"/> example_nested_branch_dag example	airflow	1	@daily	2020-10-26, 21:07:37	9	▶ ↺ 🗑️	...
<input type="checkbox"/> example_passing_params_via_test_command example	airflow		* / 1 *			▶ ↺ 🗑️	...

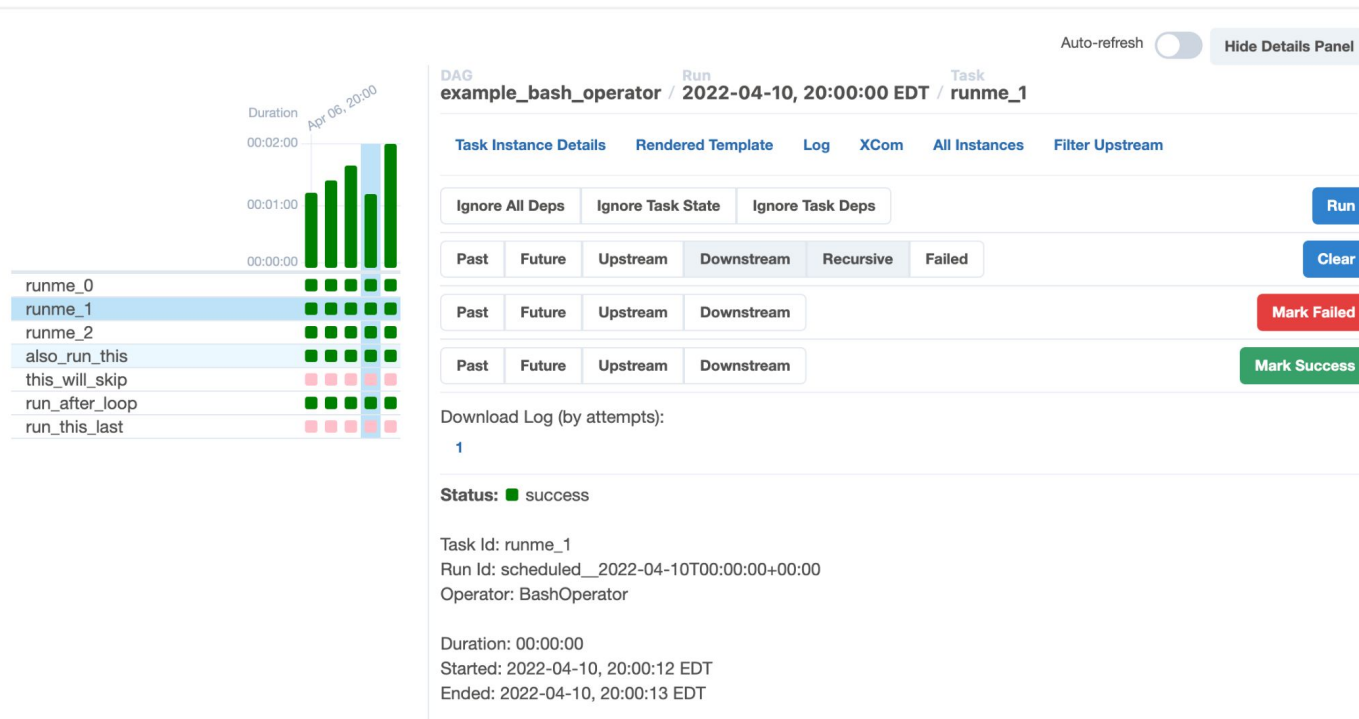


Utilisation de la console web Airflow



Utilisation de la console web Airflow

■ queued ■ running ■ success ■ failed ■ up_for_retry ■ up_for_reschedule ■ upstream_failed ■ skipped ■ scheduled ■ deferred ■ no_status



Utilisation de la console web Airflow

Airflow DAGs Security Browse Admin Docs 10:46 PDT (-07:00) FB

DAG: example_bash_operator success schedule: 0 0 ***

Tree Graph Calendar Task Duration Task Tries Landing Times Gantt Details <> Code

2021-06-02T09:27:27-05:00 Runs 25 Run manual__2021-06-02T16:27:26.797940+00:00 Layout Left > Right Find Task... Update

BashOperator DummyOperator queued running success failed up_for_retry up_for_reschedule upstream_failed skipped scheduled no_status

Auto-refresh

```
graph LR; runme_0[runme_0] --> also_run_this[also_run_this]; runme_1[runme_1] --> run_after_loop[run_after_loop]; runme_2[runme_2] --> run_after_loop; runme_2 --> this_will_skip[this_will_skip]; also_run_this --> run_after_loop; this_will_skip --> run_after_loop; run_after_loop --> run_this_last[run_this_last];
```



Utilisation de la console web Airflow

The screenshot displays the Apache Airflow web interface. In the background, the 'DAG: example_bash' is visible with a 'Tree View' and 'Graph View' toggle. The 'Task Instance' modal is open, showing details for the task 'run_after_loop' at '2020-10-01T00:00:00+00:00'. The modal includes tabs for 'Task Instance Details', 'Rendered', 'Task Instances', 'View Log', and 'Filter Upstream'. Under 'Task Actions', there are buttons for 'Ignore All Deps', 'Ignore Task State', 'Ignore Task Deps', and 'Run'. Below these are filters for 'Past', 'Future', 'Upstream', 'Downstream', 'Recursive', and 'Failed', with a 'Clear' button. Further down are buttons for 'Mark Failed' and 'Mark Success'. A 'Close' button is at the bottom right. The background interface also shows a 'DAG: example_bash' with a 'Tree View' and 'Graph View' toggle, a 'Run' button, and a 'Find Task...' search bar.

Task Instance: run_after_loop
at: 2020-10-01T00:00:00+00:00

Task Instance Details | Rendered | Task Instances | View Log | Filter Upstream

Task Actions

Ignore All Deps | Ignore Task State | Ignore Task Deps | Run

Past | Future | Upstream | Downstream | Recursive | Failed | Clear

Past | Future | Upstream | Downstream | Mark Failed

Past | Future | Upstream | Downstream | Mark Success

Close



Utilisation de la console web Airflow



Airflow

DAGs

Datasets

Security

Browse

Admin

Docs

10:28 UTC

TA

Datasets

URI

s3://dag1/output_1.txt

s3://unrelated_task/dataset_other_unknown.txt

s3://consuming_2_task/dataset_other_unknown.txt

s3://dag2/output_1.txt

s3://consuming_1_task/dataset_other.txt

s3://unrelated/dataset3.txt

s3://unrelated/dataset_other_unknown.txt

s3://this-dataset-doesnt-get-triggered



Comment créer et démarrer des dags

- Exemples de dags : https://github.com/apache/airflow/tree/main/airflow/example_dags
- Bien lire la documentation. Airflow a mis en place une très bonne documentation :

<https://airflow.apache.org/docs/apache-airflow/2.4.3/index.html>



01 — Avancé

01 — Utilisation des variables/connections dans les DAGs

02 — Planifier les DAGs

03 — Définir les dépendances entre les dags

04 — Création d'Opérateur, Capteur, Hook



Utilisation des variables/connections dans les dags

List Variable			
Search ▾			
<div><div><div>+</div></div><div>Actions ▾</div><div>←</div></div>			Record Count: 6
<input type="checkbox"/>	Key ▴ ▾	Val ▴ ▾	Is Encrypted ▴ ▾
<input type="checkbox"/>	<div><div></div><div>airtable_api_key</div></div>	*****	True
<input type="checkbox"/>	<div><div></div><div>airtable_base_key</div></div>	appzasdasdasdas	True
<input type="checkbox"/>	<div><div></div><div>environment</div></div>	prod	True
<input type="checkbox"/>	<div><div></div><div>pipedrive_env</div></div>	pipedrive	True
<input type="checkbox"/>	<div><div></div><div>postgres_env</div></div>	prod	True
<input type="checkbox"/>	<div><div></div><div>snowflake_password</div></div>	*****	True



Utilisation des variables/connections dans les dags

[DAGs](#)[Security](#) ▾[Browse](#) ▾[Admin](#) ▾[Docs](#) ▾

List Connection

[Actions](#) ▾**Conn Id** ↑↓**Conn Type** ↑↓**Host** ↑↓

airflow_db

mysql

mysql



aws_default

aws



Utilisation des variables/connections dans les dags

- Variables

```
from airflow.models import Variable
foo = Variable.get("foo")
bar = Variable.get("bar", deserialize_json=True)
baz = Variable.get("baz", default_var=None)
```

- Connections

```
sensor = S3KeySensor(
    task_id='s3_file_sensor',
    bucket_key='path/to/file.csv',
    bucket_name='my_bucket',
    s3_conn_id='s3_conn',
    poke_interval=30,
    dag=dag
)
```



Planifier les DAGs

- En utilisant des CRON :
 - cron expression, ex : `schedule='5 4 * * *'` => *4:05 AM tous les jours*
 - cron pré-configuré, ex : `schedule='@hourly'` => toutes les heures
- En utilisant des "datasets" :

```
dataset1 = Dataset(f"{DATASETS_PATH}/dataset_1.txt")  
dataset2 = Dataset(f"{DATASETS_PATH}/dataset_2.txt")
```

```
with DAG(  
    dag_id='dataset_dependent_example_dag',  
    catchup=False,  
    start_date=datetime(2022, 8, 1),  
    schedule=[dataset1, dataset2],  
) as dag:
```



Définir les dépendances entre les dags

1. Utilisation de ExternalTaskSensor

```
with DAG(  
    dag_id='dag_with_ext_task_sensor',  
    catchup=False,  
    start_date=datetime(2022, 8, 1),  
    schedule='0 3 * * *',  
) as dag:
```

```
external_task_sensor = ExternalTaskSensor(  
    task_id='external_task_sensor',  
    external_task_id='end',  
    external_dag_id='dependent_dag'  
)
```

2. Utilisation de Dataset

```
with DAG(  
    dag_id='dataset_dependent_example_dag',  
    catchup=False,  
    start_date=datetime(2022, 8, 1),  
    schedule=[dataset1, dataset2],  
) as dag:
```



Création d'Opérateur, Capteur, Hook

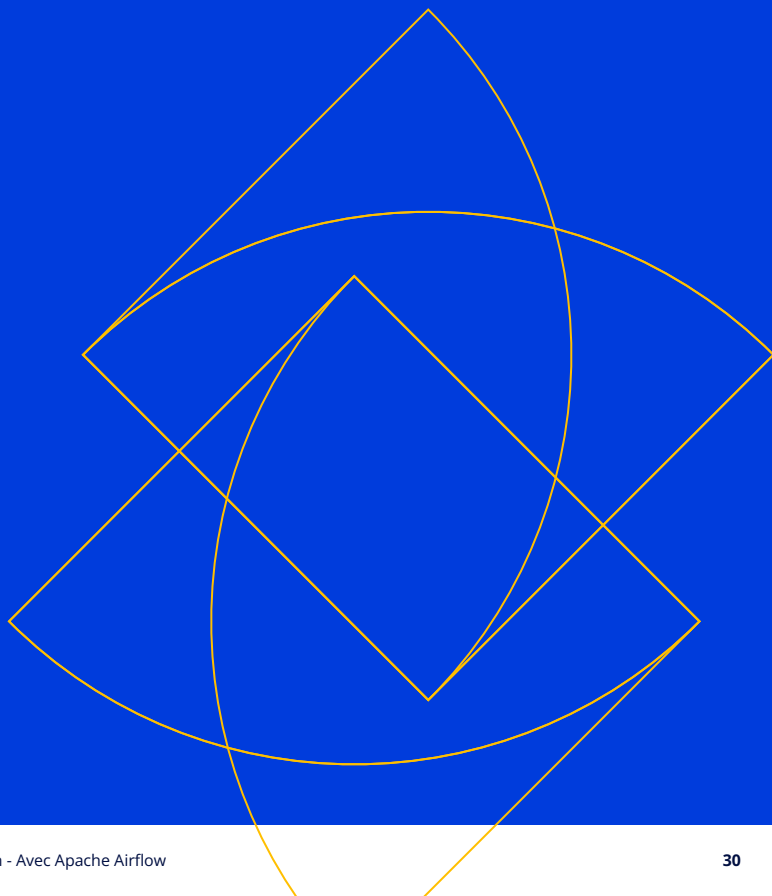
```
class LogMessageOperator(BaseOperator):

    @apply_defaults
    def __init__(self, message, *args, **kwargs):
        super(LogMessageOperator, self).__init__(*args, **kwargs)
        self.message = message

    def execute(self, context):
        current_time = datetime.now()
        self.log.info(f"[{current_time}] {self.message}")
```



02.1— Setup



Vérification des accès

- **Accès Gitlab** : <https://gitlab.ippon.fr/datacademie-group>
- **Accès Sandbox AWS** : Application Google -> Amazon Web Services -> **ippon-sandbox-00**
- **Accès Airflow (région N. Virginia)** : Amazon Managed Workflows for Apache Airflow (MWAA) -> **datacademie-airflow**



Connexion à airflow



Amazon MWAA > Environments

Airflow environments

Environments (1)

Find environments

Refresh Edit Delete Actions Create environment

Name	Status	Created date	Airflow version	Airflow UI
 datacademie-airflow	 Available	Mar 10, 2023 14:28:27 (UTC+01:00)	2.4.3	Open Airflow UI

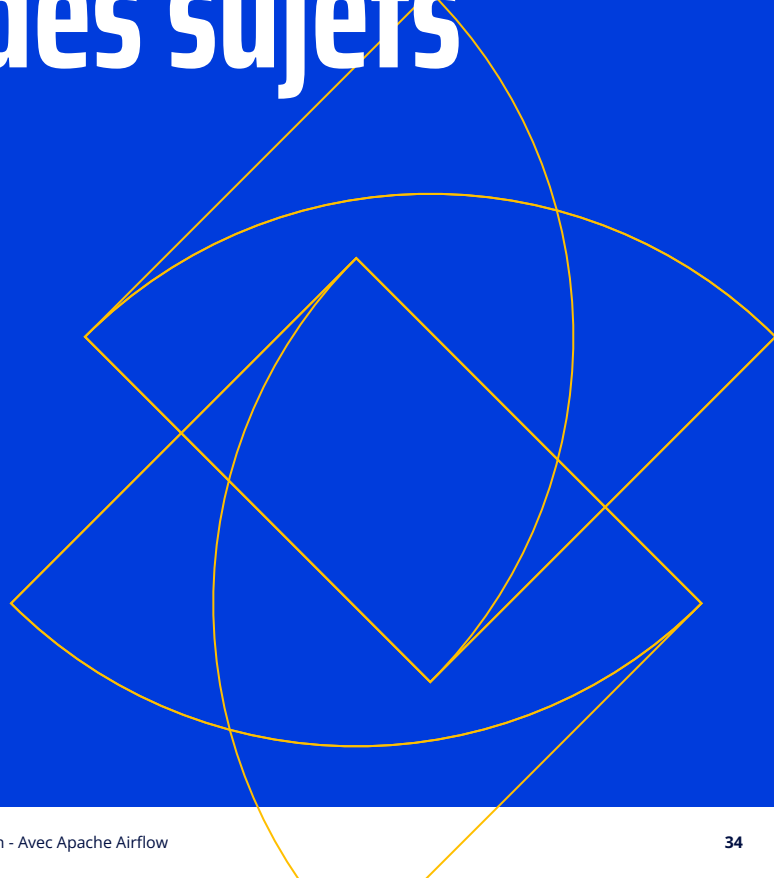


Création d'un premier dag

- Clone du projet Gitlab "datacademie-airflow"
- Créer une branche "feat/XXX"
- Créer un dossier avec votre identifiant IPPON suivant d'un sous dossier du nom de votre dag
 - ex: jgros/exemple_1/jgros_exemple_1_dag.py



02.2 — Présentation des sujets



ETL avec les lambdas

Tout est spécifié dans le README.md de l'exercice sur gitlab.

Vous devez:

- Mettre en place l'infrastructure terraform nécessaire pour l'exercice
- Créer le dag avec :
 - Les tâches pour extraire les données depuis l'api "velibmetropole"
 - Les tâches pour transformer vos données
 - Les tâches pour charger les données transformées dans Snowflake



Analytics

- Création des requêtes SQL pour générer un modèle de données statistique
- Création du dag permettant d'orchestrer la transformation des données dans Snowflake



Amélioration / Optimisation de vos DAGs

- Gestion des dépendances entre dags
- Création de variables dans vos dags
- Généralisation des tâches Snowflake / Lambda
- Création de groupe de tâche Airflow afin de grouper vos tâches

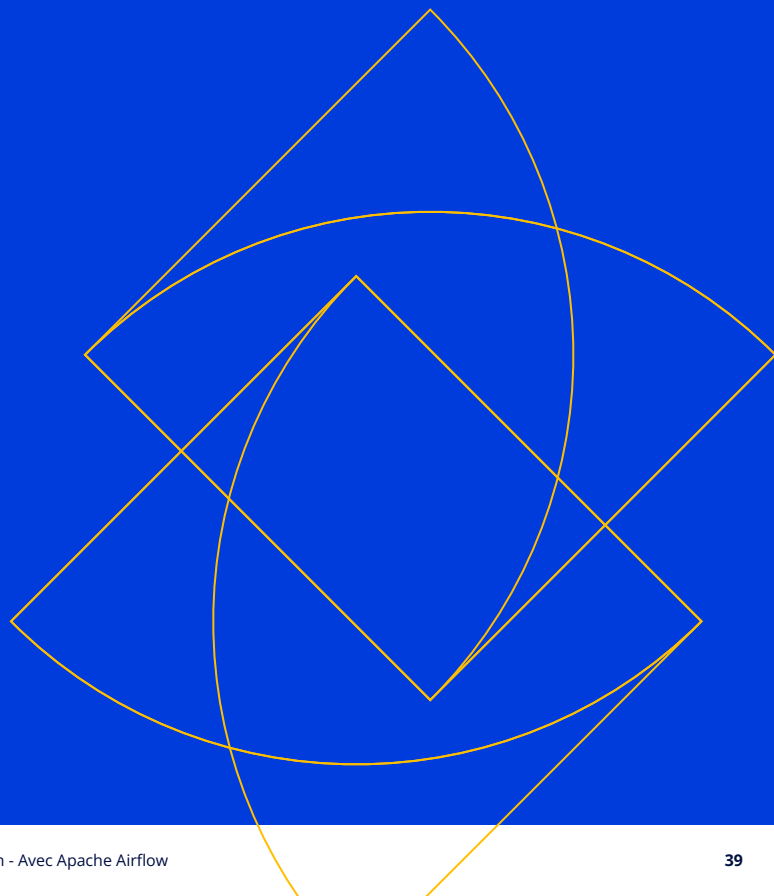


Automatisation des flux de données précédents

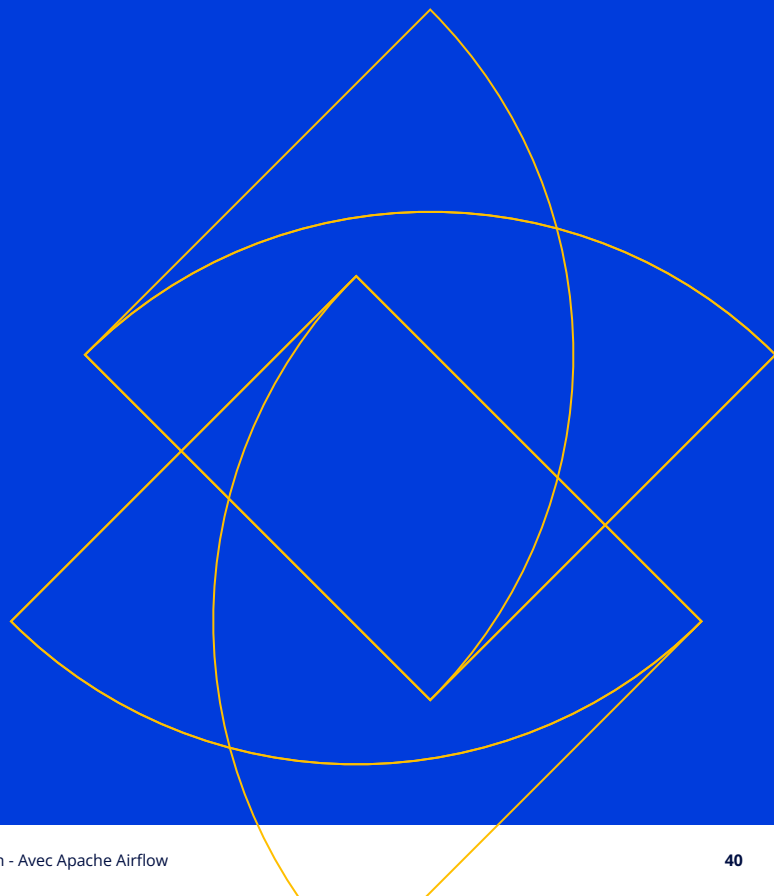
- Orchestrer l'ensembles des tâches précédentes :
 - Airbyte
 - DBT Cloud
 - Snowflake



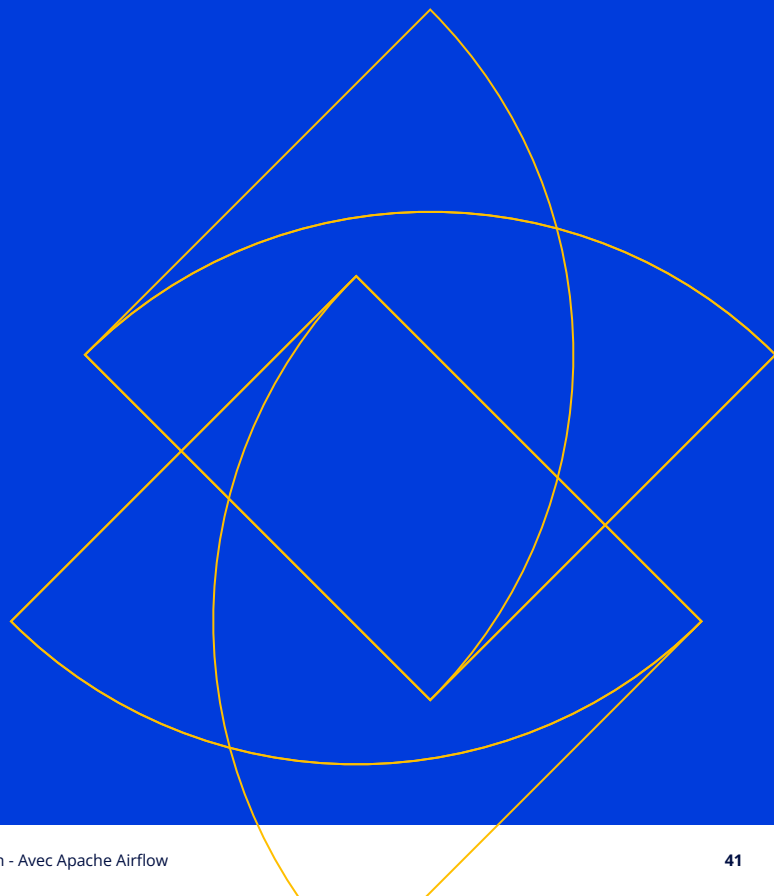
02.3 — TPs



02.4 — Conclusion



03 — ROTI





www.ippon.fr

contact@ippon.fr — +33 1 46 12 48 48 — @ipponTech

